Dear Hiring Mgr,

In this document, I will briefly explain how I tackled the test problems and what my design idea is.

Question 1 – x line overlap:

There are many overlap cases: left side overlap, right side overlap, left endpoint overlap, right endpoint overlap, contain, within. And this is logically symmetric, which means while x1 overlaps x2 on its left side, x2 overlaps x1 on its right side at the same time. So, if we enumerate all these cases in the code, there are 2\*6 = 12 judgements. But this is tedious.

However, I notice that there are only two non-overlap cases: on left side, on right side. And because it is logically symmetric, I only need to check if x1 is on x2’s left side or right side without overlapping, which makes the code neat.

Question 2 – version comparison:

First in my code I do some input checking to make sure it is valid. My consideration is:

1. A library is used widely and so we should consider all kinds of situations as much as we can.
2. String type inputs could have many kinds of errors because of its variety.

In my answers I check some errors, like no input, only one input, non-numerical input, etc. And I know I didn’t check all kinds of situations. For example, is version “1.4.0.6” acceptable, is “1-5” valid, etc. If in production we need to consider all these but in this test I just to want to demonstrate that I have think about this aspect so only part of the possible errors are checked.

My code checks both single digit case (like “1.2”) and multiple digits case (like “18.036”).

Question 3 – cache library:

This is a tough one. After reading the document, my first impression is this is an in-memory cache, which could have different modes (local or remote) and topologies (Client-Server or P2P). I think a simple but workable version can get me a better score than a complex, looks-powerful, but not working version, so I start to implement it as a local cache first.

In my understanding, the basic APIs for a local cache includes at least *start(configs)*, *get()*, *stop()*. Function *set()* should be internal but in my code I make it public just for test. The data structure I use to store cached entries is Dictionary, because its look-up performance is the best (*O(1)*). When starting the cache I also spawn a guarding thread called Ranger to monitor the states of the cache and react properly, like for cache overflow and cache expire. To keep it simple and work, in both cases I clear the cache directly, while I know that for overflow we may use more complex strategies, for example, deleting some of the oldest entries according to LRU.

A common behavior of a cache is that if a query can’t find a result in cache the query will go to the data store behind and then update the cache with the latest successful query (LRU again). But here in test I don’t have a database on hand. So I use another Dictionary to fake a database as a test stub so that I can implement the desired behavior of the cache. So a TODO here is to implement a database driver, similar to JDBC, to connect and query a database.

After the basic functionalities are done, next I consider how to make it synchronized. My idea is to use P2P fashion. That is, every cache online is a peer and there is no a centralized server. When a cache gets updated, it will also broadcast this update to other online caches. This way the data between distributed caches can be consistent. To do this, I also need functions to broadcast updates. The functions could be RESTful APIs. This is another TODO. There are many frameworks for this but obviously I won’t be able to do it in this test. Note that this way has a risk: when scaling up, the broadcast could be too frequent and thus have a big burden on the network and also impact its real-time performance.

Regarding distribution, I didn’t think it through much, because I think this could be done on the application layer or in a cache client, in which each query can be distributed to caches in different locations based on some mechanisms, like hashing. So this functionality is not necessarily in the cache library itself (if it is a cache server).

So, the above is pretty much what I can do for this question in a limited timeframe. Hopefully it can help you to understand my code.

Regarding the test:

In all my answers, I provided many tests to verify my code. You can simple execute the *\*.py* files or the *\*\_test.py* files to see the test results. Hope this helps.